Virtual Products

Virtual Product Architecture

Goal

Simplify discovery and access of virtual products by creating granules of available products on Ingest.

Traceability

EPIC: ⚠ CMR-1679 - JIRA project doesn't exist or you don't have permission to view it.

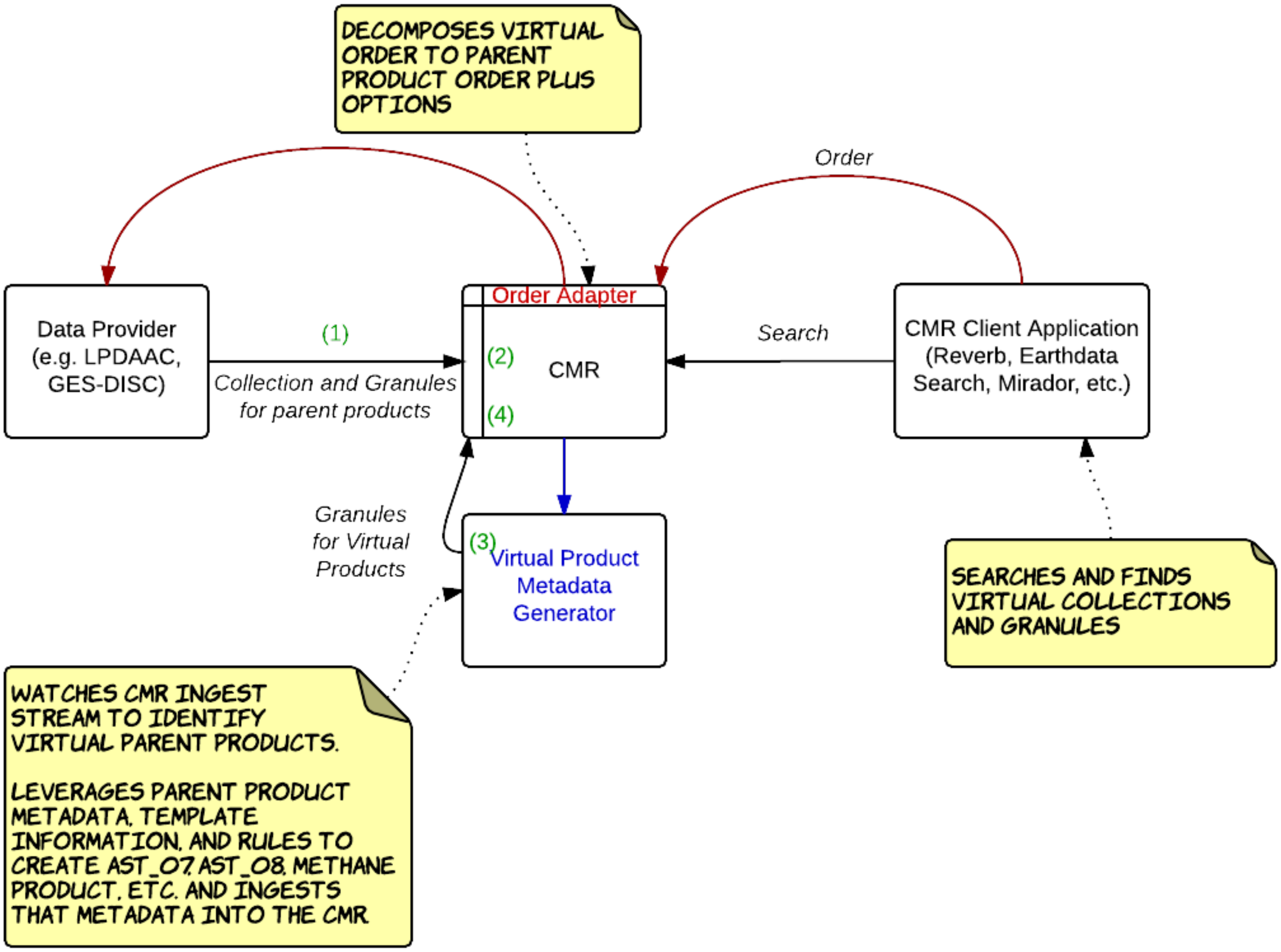
Metadata Generator

- Observe all granule and collection ingest after persistence (and potentially restrict by filter)
- Metadata generation done in parallel with indexing
- Support one granule template per on demand product (hard coded template)
- Support full lifecycle (CUD)
- Bootstrap of existing data

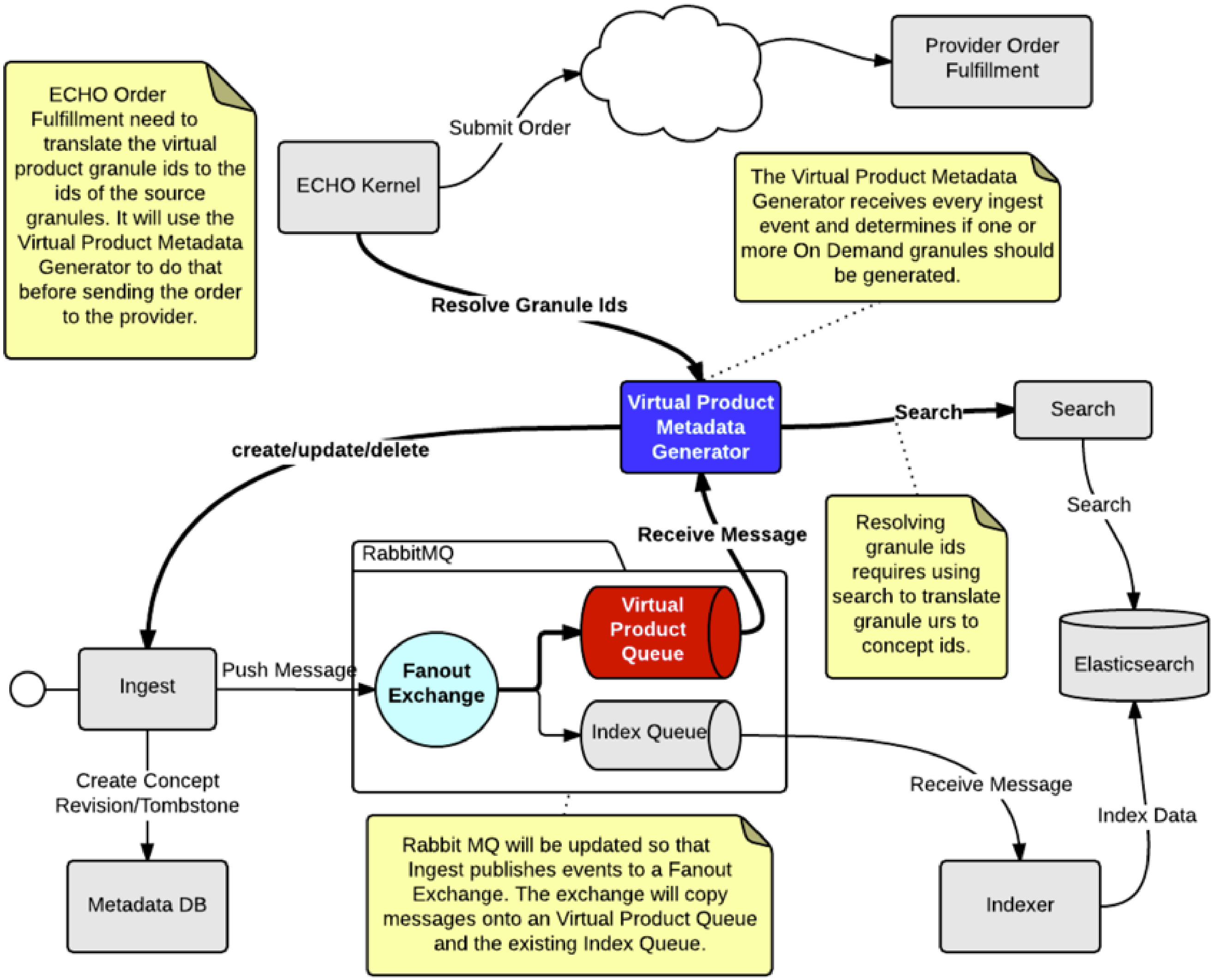
Order Adapter

- Given an order for on demand granule, determine source granule and appropriate order options needed to pass to ordering service
- Implement Order Fullfillment API
- Proxy requests to the DAACs

Associated Diagrams



DECOMPOSES VIRTUAL ORDER TO PARENT PRODUCT ORDER PLUS OPTIONS
Order
Data Provider (e.g. LPDAAC, GES-DISC)
(1)
Collection and Granules for parent products
Order Adapter
(2)
CMR
(4)
Search
CMR Client Application (Reverb, Earthdata Search, Mirador, etc.)
Granules for Virtual Products
(3)
Virtual Product Metadata Generator
SEARCHES AND FINDS VIRTUAL COLLECTIONS AND GRANULES
WATCHES CMR INGEST STREAM TO IDENTIFY VIRTUAL PARENT PRODUCTS.
LEVERAGES PARENT PRODUCT METADATA, TEMPLATE INFORMATION, AND RULES TO CREATE AST_07, AST_08, METHANE PRODUCT, ETC. AND INGESTS THAT METADATA INTO THE CMR


ECHO Order Fulfillment need to translate the virtual product granule ids to the ids of the source granules. It will use the Virtual Product Metadata Generator to do that before sending the order to the provider.
ECHO Kernel
Submit Order
Provider Order Fulfillment
The Virtual Product Metadata Generator receives every ingest event and determines if one or more On Demand granules should be generated.
Resolve Granule Ids
Virtual Product Metadata Generator
Search
Search
Search
create/update/delete
Receive Message
Resolving granule ids requires using search to translate granule urs to concept ids.
RabbitMQ
Virtual Product Queue
Fanout Exchange
Index Queue
Elasticsearch
Ingest
Push Message
Create Concept Revision/Tombstone
Metadata DB
Receive Message
Index Data
Indexer
Rabbit MQ will be updated so that Ingest publishes events to a Fanout Exchange. The exchange will copy messages onto an Virtual Product Queue and the existing Index Queue.

Design Decisions and Assumptions

- Start with FRB issue for LP(July)
- GES DISC has very little interest in providing ISO granules, need to support ECHO 10 granule templates.
- On Demand Granule metadata should be created and ingested just like normal granules
- Templates should be provided for granules only as providers will want to manually control the descriptions, etc for their collections.
- Providers will need to manually create On Demand collections and associated forms prior to us creating granules.
- AIRES and ASTERI1T
    - drive growth and force us to buy new HW
    - need to determine how to manage elastic instances for large collections
- Template Format is up to us. We will initially just get the logic/rules from the provider in a non-codeish format:
    - Start with existing granule. Hardcode anything that can hard code.
    - Copy spatial and temporal bounds from source granule
    - Online access URL is a derivative from source granule

On Demand Metadata Generator Design and Estimates

- Ingest Broadcast of events
    - We need to change ingest so that messages are delivered to the indexer queue and the on demand generation queue
    - Rabbit MQ Fanout exchanges as described here look like the right way to do that.
    - Need to test message confirms. Message must be placed on *all* queues in order to be considered successful
    - Tasks
        - Understand/prototype the ways we can handle broadcast of events. (2)
- Setting up new Microservice (2)

  - Communicating with SAs, verifying it's deployed, Setting up the project, etc.
- Configuration
  - We need to allow configuration of On Demand Collections.
    - Matching granule Configuration (1.5)
      - Source collection - Specified via entry title or entry id
      - Target Collection
      - Granule metadata filters
        - We need to be able to specify specific requirements of the granule metadata that will limit which granules are generated from the source granules.
        - Example: AST_07XT on demand granules should be created if all of these are true
          - additional attribute SWIR_ObservationMode = "ON"
          - additional attribute VNIR1_ObservationMode = "ON"
          - additional attribute VNIR2_ObservationMode = "ON"
          - DayNightFlag = "DAY"
    - Template (2)
      - need to design this and implement
      - target granule ur = source granule ur + "_" + collection short name
      - Q: What about the max length of the granule ur?
- New Granule Generation
  - Creations

    Filter granule events to the ones that match any of the configured on demand collections (1)
      - match collection, fetch metadata from mdb, match on granule filters

    Generate new granule events from source granule events (0.5)
      - Use template to generate XML
      - Process events by sending them to ingest as PUT
  - Updates
    - If they update a granule such that it doesn't match and previously did then we need to detect that and generate a corresponding delete (2)
      - Previous revision exists
        - Fetch previous revision (if revision > 1)
        - See if the previous revision matched the granule filter.If previous matches and the new one doesn't then generate a deletePrevious revision doesn't exist
      - Fetch previous revision (if revision > 1) and it returns 404
        - Generate a delete for any non-matching filter
  - Deletes (1)
    - Always generate deletes for every target on demand collection. Some of the granules may not exist.
    - Granule filters would not be used during this case.
  - Process events by sending them to ingest as Delete
  - Failure cases (1)
    - Need wait queues in place for failures
    - Failure should have a retry mechanism similar to the indexer now with multiple wait queues.

Bootstrap Design and Estimates

- Design
  - Shouldn't go through ingest API. We need it to be fast
  - Build into bootstrap since it will need a direct dependency on metadata db and indexer
  - Use core async and connect things together
- Core.async flow similar to db synchronization
  - API to call this (0.5)
    - design, implement, document
  - Step 1 (0.25)
    - The list of configured source collections
    - Pull granules in chunks from the database for each source collections
      - This can pull all revisions.
  - Step 2 (0.25)
    - Convert each source granule in a chunk to one or more demand granule
  - Step 3 (0.5)
    - Validate each on demand granule
      - Might not need to do this. Maybe just use umm validation.
      - We'll need the parent collection to use umm validation. That could be passed through from step 1 to avoid constantly refetching it.
    - Save each on demand granule to metadata db
      - Revision conflict will be 409. Consider this processed at this point because a later revision was received.
    - Index each on demand granule
  - Integration Testing (1.5)
  - Workload Performance Testing (1.5)
    - Test how long it takes and makes sure it's successful

Order Fulfillment Design and Estimates

ECHO will get an order with a set of order items. Each order item contains a dataset id, granule ur, and catalog item guid. I'll refer to that as the granule id triple. ECHO will need to make a request to the CMR with all the granule id triples in the order. The CMR will respond with a new set of granule ids. Most of the time they'll be the same. When some of the granules are on demand granules the CMR will have converted them to their source granule ids. ECHO should then take those new ids and replace the ones in the order to send to the provider.
Here's that in step form.
ECHO Order Submission Extra Steps
1. Gather the granule ur, dataset id, and catalog item guid of all the order items. (You should have them. You're about to send them to the provider)
2. Make an HTTP request with those IDs as JSON to the CMR.
3. Parse the response (JSON) which will contain a list of granule ur, dataset id, and catalog item guid triples.
4. Replace the IDs in the order with the ids in the response. (The index of the id will tell you which order item to set it in.)
This would probably be done in gov.nasa.echo.kernel.service.order.routing.orderfulfillment.v9.OrderFulfillmentClientImpl.

- CMR: Implement API taking Triples of Granule UR, Dataset Id, and Catalog Item GUID (1.75)
    - Determine if any match on demand collections and convert
        - DataSetId to entry title of source collection
        - GranuleUr to granule ur of equivalent granule in the source collection
        - CatalogItemGuid to the catalog item guid of equivalent granule in the source collection
        - Conversion will require executing a search to find the equivalent granules.
- ECHO: Updated to call Order Item Id Convert through HTTP API for ever order fulfillment
    - Needs CMR endpoint configuration (0.5)
    - Update code to call CMR API (2)
    - Integration Testing (0.5)

Outstanding Questions

☑ Dan Pilone Do we need to support ordering for FRB or are those all done via Online Access?

☑ How do we generate an on demand granule ur from a source granule? It needs to be done in a way that allows extraction of the source granule ur. source granule ur + "_" + collection short name could work but would generate granule urs that exceed length limitations.

Error rendering macro 'pageapproval' : null